PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-069082
(43)Date of publication of application : 11.03.1997

(51)Int.Cl. G06F 15/16

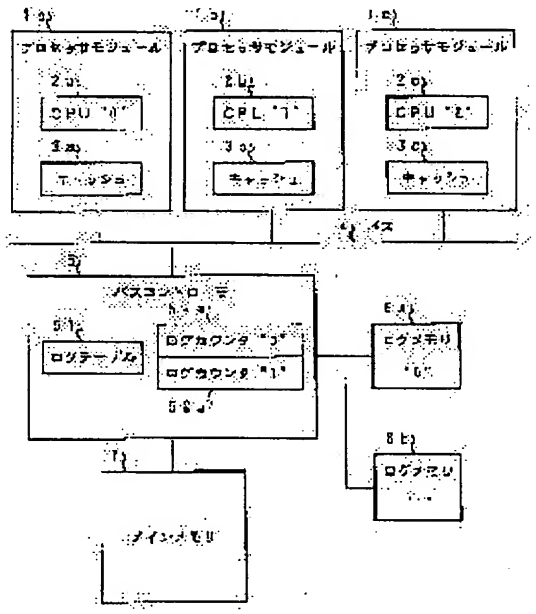
(21)Application number : 07-341839
(22)Date of filing : 27.12.1995

(71)Applicant : TOSHIBA CORP
(72)Inventor : HOSHINA SATOSHI
SAKAI HIROSHI
HIRAYAMA HIDEAKI
OMORI TAKASHI
MASUBUCHI YOSHIO
FUJII TAKAHIRO

(30)Priority
Priority number : 07151735 Priority date : 19.06.1995 Priority country : JP

(54) MULTIPRO CESSOR SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide the computer system which can greatly decrease stand-by states in the whole system at check point acquisition.
SOLUTION: This system is equipped with a plurality of log memories (log memories 6a-6b) which hold the update history of a main memory 7 and CPUs 2a-2c record the update history of the main memory 7 in one of the log memories 6a-6b; when the internal state and the contents of a cache memory are written out to the main memory 7 in response to check point acquisition, switching is so performed that CPUs having finished writing them out record the update history of the main memory 7 in another log memory, thereby restarting ordinary data processing without waiting for other CPUs to have acquired check points.



LEGAL STATUS

[Date of request for examination] 06.03.1997
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number] 3085899
[Date of registration] 07.07.2000
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

BEST AVAILABLE COPY

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－69082

(43)公開日　平成９年(1997)３月11日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　15/16	４３０		Ｇ０６Ｆ　15/16	４３０Ｂ	

審査請求　未請求　請求項の数17　ＯＬ　（全 21 頁）

(21)出願番号　特願平7－341839

(22)出願日　平成７年(1995)12月27日

(31)優先権主張番号　特願平7－151735

(32)優先日　平７(1995)６月19日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　保科　聡
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

(72)発明者　酒井　浩
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

(72)発明者　平山　秀昭
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

(74)代理人　弁理士　鈴江　武彦

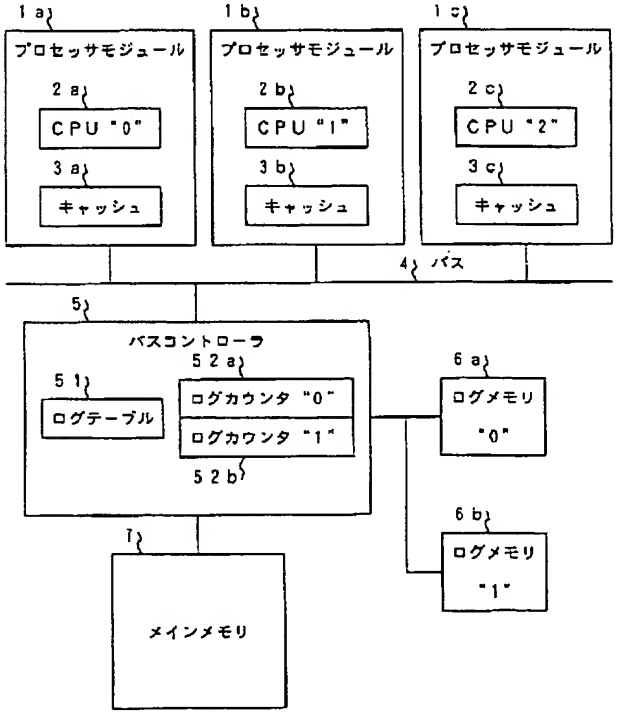
最終頁に続く

(54)【発明の名称】　マルチプロセッサシステム

(57)【要約】

【目的】チェックポイント取得時におけるシステム全体の待機状態を大幅に減少させることを可能とする計算機システムを提供する。

【構成】メインメモリ７の更新履歴を保持するログメモリを複数備え（ログメモリ６ａ～６ｂ）、ＣＰＵ２ａ～２ｃがその複数のログメモリ６ａ～６ｂの中のいずれか一つにメインメモリ７の更新履歴を記録していき、チェックポイント取得に伴ってその内部状態およびキャッシュメモリの内容をメインメモリ７に書き出す際に、その書き出しを完了したＣＰＵから他のいずれかのログメモリにメインメモリの更新履歴を記録するように切り替えて、他のＣＰＵのチェックポイント取得完了を待機せずに通常のデータ処理を再開することを特徴とする。



1

【特許請求の範囲】

【請求項１】　キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、

前記共有メモリの更新履歴を示す履歴情報を格納する第１および第２のログメモリと、

これら第１および第２のログメモリの中で履歴情報の格納に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段とを具備し、

前記各プロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第１および第２のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項２】　前記複数のプロセッサの中の少なくとも１つのプロセッサのキャッシュメモリのライトポリシーをライトバックからライトスルーに変更し、そのライトスルーキャッシュを持つプロセッサにリアルタイムプロセスを実行させる手段をさらに具備し、

前記ライトスルーキャッシュを持つプロセッサは、前記チェックポイントにおいてその内部状態だけを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第１および第２のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したリアルタイムプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項３】　前記第１および第２のログメモリは、

１つの物理メモリと、

前記第１のログメモリにおける更新履歴格納番地を指定するポインタ値を保持する第１のカウンタであって、前記物理メモリの先頭番地から最終番地に向かって前記ポインタ値が順次インクリメントされる第１のカウンタと、

前記第２のログメモリにおける更新履歴格納番地を指定

2

するポインタ値を保持する第２のカウンタであって、前記物理メモリの最終番地から先頭番地に向かって前記ポインタ値が順次デクリメントされる第２のカウンタとを具備することを特徴とする請求項１記載のマルチプロセッサシステム。

【請求項４】　前記ログメモリ選択手段は、

前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、

前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応する前記第１および第２のカウンタの一方のポインタ値を選択する手段とを具備することを特徴とする請求項３記載のマルチプロセッサシステム。

【請求項５】　前記第１および第２のログメモリは、

前記履歴情報を格納する第１フィールドと前記共有メモリにデータ書き込みを行ったプロセッサが現在使用しているログメモリを示すログメモリ識別子を格納する第２フィールドとを各々が含む複数のエントリを有する１つの物理メモリを具備し、

前記各プロセッサによるログメモリの切り替えに応答して、前記第２フィールドに格納されるログメモリ識別子の値が変更されることを特徴とする請求項１記載のマルチプロセッサシステム。

【請求項６】　前記ログメモリ選択手段は、

前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、

前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応するログメモリ識別子を、前記書き込みに対応する履歴情報を格納すべき前記物理メモリのエントリの第２フィールドに格納する手段とを具備することを特徴とする請求項５記載のマルチプロセッサシステム。

【請求項７】　前記第１および第２の双方のログメモリにおける履歴情報格納番地を指定するポインタ値を保持する第１のカウンタであって、前記物理メモリの先頭エントリから最終エントリに向かって前記ポインタ値が順次インクリメントされる第１のカウンタと、

前記複数のプロセッサの中で最初にログメモリを切り替えたプロセッサが使用を開始した前記物理メモリの最初のエントリを示すポインタ値を保持する第２のカウンタとをさらに具備し、

前記マルチプロセッサの障害発生時は、前記第２のカウンタのポインタ値が示すエントリから前記第１のカウンタのポインタ値が示すエントリの中で、前記第１のカウ

ンタのポインタ値が示すエントリの中に格納されているログメモリ識別子と同一のログメモリ識別子が格納されているエントリの履歴情報が前記共有メモリの復元のために使用されることを特徴とする請求項５記載のマルチプロセッサシステム。

【請求項８】　キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、

前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、

前記共有メモリの更新履歴を格納する第１および第２のログメモリと、

これら第１および第２のログメモリの中で更新履歴の保存に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段と、

前記複数のプロセッサそれぞれのキャッシュメモリのライトポリシーをライトバックまたはライトスルーに設定する手段であって、前記マルチプロセッサシステムの動作状態を監視し、そのシステムのスループットとレスポンスタイムとに基づいて、ライトスルーキャッシュとして動作するキャッシュメモリを持つプロセッサ数を動的に変更する手段とを具備し、

前記ライトバックキャッシュとして動作するキャッシュメモリを持つプロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第１および第２のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開し、

前記ライトスルーキャッシュとして動作するキャッシュメモリを持つプロセッサは、その内部状態だけを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第１および第２のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項９】　前記第１および第２のログメモリは、

１つの物理メモリと、

前記第１のログメモリにおける更新履歴格納番地を指定するポインタ値を保持する第１のカウンタであって、前記物理メモリの先頭番地から最終番地に向かって前記ポインタ値が順次インクリメントされる第１のカウンタと、

前記第２のログメモリにおける更新履歴格納番地を指定するポインタ値を保持する第２のカウンタであって、前記物理メモリの最終番地から先頭番地に向かって前記ポインタ値が順次デクリメントされる第２のカウンタとを具備することを特徴とする請求項８記載のマルチプロセッサシステム。

【請求項１０】　前記ログメモリ選択手段は、

前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、

前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応する前記第１および第２のカウンタの一方のポインタ値を選択する手段とを具備することを特徴とする請求項９記載のマルチプロセッサシステム。

【請求項１１】　前記第１および第２のログメモリは、

前記履歴情報を格納する第１フィールドと前記共有メモリにデータ書き込みを行ったプロセッサが現在使用しているログメモリを示すログメモリ識別子を格納する第２フィールドとを各々が含む複数のエントリを有する１つの物理メモリを具備し、

前記各プロセッサによるログメモリの切り替えに応答して、前記第２フィールドに格納されるログメモリ識別子の値が変更されることを特徴とする請求項８記載のマルチプロセッサシステム。

【請求項１２】　前記ログメモリ選択手段は、

前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、

前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応するログメモリ識別子を、前記書き込みに対応する履歴情報を格納すべき前記物理メモリのエントリの第２フィールドに格納する手段とを具備することを特徴とする請求項１１記載のマルチプロセッサシステム。

【請求項１３】　前記第１および第２の双方のログメモリにおける履歴情報格納番地を指定するポインタ値を保持する第１のカウンタであって、前記物理メモリの先頭エントリから最終エントリに向かって前記ポインタ値が順次インクリメントされる第１のカウンタと、

前記複数のプロセッサの中で最初にログメモリを切り替えたプロセッサが使用を開始した前記物理メモリの最初

5

のエントリを示すポインタ値を保持する第２のカウンタとをさらに具備し、

前記マルチプロセッサの障害発生時は、前記第２のカウンタのポインタ値が示すエントリから前記第１のカウンタのポインタ値が示すエントリの中で、前記第１のカウンタのポインタ値が示すエントリの中に格納されているログメモリ識別子と同一のログメモリ識別子が格納されているエントリの履歴情報が前記共有メモリの復元のために使用されることを特徴とする請求項１１記載のマルチプロセッサシステム。

【請求項１４】　それぞれにキャッシュメモリを備えたＣＰＵを複数有し、それらのＣＰＵがメインメモリを共有するマルチプロセッサシステムであって、上記メインメモリの更新時にそのアドレスと更新前の内容とを更新履歴として保持するログメモリを備えるとともに、上記各ＣＰＵが同期して所定のタイミングで周期的にその内部状態および必要に応じてキャッシュメモリの内容を上記メインメモリに書き出すことにより、上記マルチプロセッサシステムに障害が発生したときに、上記ログメモリに保持された更新履歴により上記メインメモリを所定の時点に復元して、その障害発生前の状態から、再実行することを実現するマルチプロセッサシステムにおいて、

上記ログメモリを複数備え、

上記各ＣＰＵがその複数のログメモリの中のいずれか一つに上記メインメモリの更新履歴を記録していき、上記特定のタイミングでその内部状態および必要に応じてキャッシュメモリの内容を上記メインメモリに書き出す際に、その書き出しを完了したＣＰＵは、他のいずれかのログメモリにメインメモリの更新履歴を記録するように切り替えて、即座に通常のデータ処理を再開することを特徴とするマルチプロセッサシステム。

【請求項１５】　ＣＰＵから書き込み要求があったときに、即座にメインメモリに書き込むライトスルー方式のキャッシュメモリを備えたＣＰＵと、一旦キャッシュメモリ内に蓄えて適宜メインメモリに書き込むライトバック方式のキャッシュメモリを備えたＣＰＵとを上記マルチプロセッサシステムに混在させ、常に所定の時間内に終了させなければならない処理を上記ライトスルー方式のキャッシュメモリを備えたＣＰＵにて実行することを特徴とする請求項１４記載のマルチプロセッサシステム。

【請求項１６】　上記各ＣＰＵの備えるキャッシュメモリは、ライトスルー方式およびライトバック方式のいずれの方式で動作するのか上記マルチプロセッサシステム上で稼働するプログラムの指令により動的に切り替えることを特徴とする請求項１５記載のマルチプロセッサシステム。

【請求項１７】　上記マルチプロセッサシステム上で稼働するプログラムは、システムの稼働状態を監視して、

6

実行されるプログラムのスループットおよび応答性を測定し、ライトスルー方式およびライトバック方式それぞれ適切な数を決定する手段を含むことを特徴とする請求項１６記載のマルチプロセッサシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は共有メモリを有するマルチプロセッサシステムに関し、特にログメモリを使用したチェックポイントリスタート方法を採用したマルチプロセッサシステムに関する。

【０００２】

【従来の技術】一般に、フォールトトレラントコンピュータシステムの構成法のひとつとして、チェックポイントリスタート方法を用いたシステム回復技術が知られている。このチェックポイントリスタート方法では、プロセスの再実行に必要な情報が定期的に保存される。この情報を保存する時点をチェックポイントと呼び、その情報を保存することをチェックポイントの取得と呼ぶ。コンピュータシステムの障害発生によってプログラム実行が中断されたとき、その中断されたプロセスの状態は障害発生前のチェックポイントまでロールバックされ、そのチェックポイントから再実行される。

【０００３】チェックポイント取得処理で保存される情報には、プロセッサの内部状態、キャッシュメモリ、メインメモリ等がある。最近では、チェックポイント取得処理に要する時間、つまり通常のプログラム実行が中断される時間を短くするために、ログメモリを使用したチェックポイントリスタート方法が採用され始めている。

【０００４】この方法では、あるチェックポイントから次のチェックポイントまでの期間において、プロセス実行に伴ってメインメモリが更新される度にその更新前データなどが更新履歴情報としてログメモリに採取される。コンピュータシステムに障害が発生したとき、ログメモリの内容によってメインメモリは障害発生前のチェックポイントの時点に復元される。従って、ログメモリを使用したチェックポイントリスタート方法を採用した場合には、チェックポイント取得処理で保存しなければならない情報はプロセッサの内部状態とキャッシュメモリだけとなり、チェックポイント取得処理に要する時間を比較的短くすることができる。このようなログメモリを使用したチェックポイントリスタート方法を採用したコンピュータシステムの典型的な構成例を図１８に示す。

【０００５】このシステムは、プロセッサモジュール１００、バス４００、バスコントローラ５００、ログメモリ６００、およびメインメモリ７００を備えている。プロセッサモジュール１００は、ＣＰＵとキャッシュメモリを備えている。

【０００６】ＣＰＵがメインメモリ７００上のデータを更新する時、バスコントローラ５００は、その更新履歴

7

をログメモリ６００に記録する。このログメモリ６００の残記憶容量がある一定サイズを下回ったとき、ＣＰＵは、チェックポイント取得処理を開始し、その内部状態とキャッシュメモリの内容をメインメモリ７００に保存する。このチェックポイント取得処理が完了した時点でログメモリ６００に保存された更新履歴情報は不要となる。そして、ログメモリ６００が初期化されてその更新履歴情報が破棄された後、ＣＰＵによって再び通常のプログラム実行処理が再開される。通常のプログラム実行処理が再開されてから次のチェックポイントまでの期間におけるメインメモリ７００の更新履歴は、ログメモリ６００に保存される。

【０００７】

【発明が解決しようとする課題】しかしながら、このようなログメモリを使用したチェックポイントリスタート方法を、メインメモリ７００を共有する複数のプロセッサモジュールを含むマルチプロセッサシステムに採用すると、各プロセッサは、自身のチェックポイント取得処理を完了しても、他の全てのプロセッサがチェックポイント取得処理を完了するまで、通常のプログラム実行処理を再開できないという問題が生じる。以下、マルチプロセッサシステムにおけるチェックポイント取得処理に関する問題点を図１９を参照して詳述する。

【０００８】ここでは、３つのＣＰＵ、ＣＰＵ“０”、ＣＰＵ“１”、ＣＰＵ“２”がメインメモリ７００を共有するマルチプロセッサシステムを例にとって説明する。

【０００９】いま、これらＣＰＵ“０”、ＣＰＵ“１”、ＣＰＵ“２”は、それぞれ通常のプログラム実行処理を行なっているものとする（図１９の（１））。その後、ログメモリの残容量が予め設定された所定量を下回ったとき（図１９の（２））、ＣＰＵ“０”、ＣＰＵ“１”、ＣＰＵ“２”それぞれは、その旨を検知してチェックポイント取得処理を開始する（図１９の（３））。

【００１０】このチェックポイント取得処理の開始タイミングおよび完了までに要する時間は、その検知タイミングや、検知したときに処理中のプログラムの種類、キャッシュメモリの状態などにより異なる。このため、チェックポイント取得処理を完了するタイミングは、ＣＰＵ毎に異なる。図１９では、ＣＰＵ“０”、ＣＰＵ“２”、ＣＰＵ“１”の順でチェックポイント取得処理が完了される例が示されている。

【００１１】この場合、ＣＰＵ“１”のチェックポイント取得処理が完了するまでは、ＣＰＵ“０”、ＣＰＵ“２”は通常のプログラム処理を再開することができず、ウエイト状態となる（図１９の（４））。なぜなら、全てのＣＰＵのチェックポイント取得処理が完了する時点（図１９の（５））までは、障害発生に備えて、ログメモリ６００に記録された更新履歴をそのまま保持

8

しておく必要があるためである。もし全てのＣＰＵのチェックポイント取得処理が完了する前にあるＣＰＵが通常のプログラム実行処理を再開してしまったならば、そのプログラム実行処理に伴うメモリ更新によって、ログメモリ６００の内容が書き換えられてしまう。この場合、もし障害が発生すると、その障害発生前のチェックポイントの時点にメインメモリを正常に復元することはできなくなる。

【００１２】従って、ＣＰＵ“０”、ＣＰＵ“１”、ＣＰＵ“２”は、それら全てのＣＰＵのチェックポイント取得処理が完了した後に、通常のプログラム実行処理を再開する（図１９の（６））。

【００１３】このように、マルチプロセッサシステムにおいては、各プロセッサは、自身のチェックポイント取得処理を完了しても、他の全てのプロセッサがチェックポイント取得処理を完了するまで、通常のプログラム実行処理を再開できないという問題がある。また、最近では、キャッシュメモリの大容量化等により、各プロセッサのチェックポイント取得処理そのものに要する時間も長くなる傾向にある。本発明は上記実情に鑑みてなされたものであり、各プロセッサが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようにし、システム全体の待機時間を大幅に減少させることができるマルチプロセッサシステムを提供することを目的とする。

【００１４】

【課題を解決するための手段】本発明は、キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、前記共有メモリの更新履歴を示す履歴情報を格納する第１および第２のログメモリと、これら第１および第２のログメモリの中で履歴情報の格納に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段とを具備し、前記各プロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第１および第２のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とする。

9

【００１５】このマルチプロセッサシステムにおいては、２つのログメモリが設けられており、プロセッサ毎にそれらログメモリを切り替えて使用することができる。このため、チェックポイント取得処理を完了したプロセッサの順で、使用するログメモリをそれまで使用していたログメモリから他方のログメモリに切り替えることにより、それまで使用していたログメモリの内容を破壊することなく、通常のプログラム実行処理を再開することができる。よって、各プロセッサが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようになり、システム全体の待機時間を大幅に減少させることができる。２つのログメモリは、１つの物理メモリ上に論理的に実現することが好ましい。これにより、ハードウェアの追加を招くことなく、低コストで高性能のシステムを実現できる。

【００１６】また、一部のプロセッサに備えられるキャッシュメモリをライトスルー方式にすることにより、そのプロセッサのチェックポイント取得時間を大幅に削減でき、たとえば、個々の処理についてのレスポンスは多少悪化させることとなっても、常に所定の時間内に終了させる必要がある処理（リアルタイム処理）などをこのプロセッサで処理させるようにすれば、チェックポイント取得に伴う弊害を回避することが可能となる。

【００１７】なお、ここでいうライトスルー方式のキャッシュメモリとは、プロセッサから書き込み要求があったときに、即座にメインメモリに書き込みを行なうタイプのキャッシュメモリをいい、また、プロセッサから書き込み要求があったときに、一旦キャッシュメモリ内に蓄えて、必要に応じてメインメモリに書き込みを行なうタイプのキャッシュメモリをライトバック方式のキャッシュメモリという。

【００１８】チェックポイント取得に費やす時間を分析調査すると、キャッシュメモリ内の更新データのメインメモリへの書き戻し、すなわちキャッシュフラッシュに費やす時間が大部分を占めることがわかる。したがって、このライトスルー方式のキャッシュメモリを備えたプロセッサのチェックポイント処理は、ライトバック方式のキャッシュメモリを備えたプロセッサのチェックポイント処理と比較して、非常に短時間で終了することがわかる。

【００１９】すなわち、このライトスルー方式のキャッシュメモリとライトバック方式のキャッシュメモリとを適切な数でシステム内に混在させることにより、バストラフィックを増やすことなく、且つ全体のパフォーマンスを低下させずに、リアルタイム処理を実行することができる。

【００２０】

【発明の実施の形態】

（第１実施形態）図１はこの発明の第１の実施形態に係わるマルチプロセッサシステムの構成を示す図である。

10

【００２１】図１に示されているように、このマルチプロセッサシステムは、メインメモリ７を共有する３つのプロセッサモジュール１ａ～１ｃを備えている。これらプロセッサモジュール１ａ～１ｃはバス４に接続されており、それぞれＣＰＵ２ａ～２ｃと、キャッシュメモリ３ａ～３ｃとから構成されている。これらキャッシュメモリ３ａ～３ｃは、それぞれＣＰＵ２ａ～２ｃの一次キャッシュ、または二次キャッシュとして動作する。各プロセッサモジュールでは、ＣＰＵがメインメモリ７からデータの読み出しを要求した際に、まずキャッシュメモリの中に該当するデータが保持されているかどうかを検査する。ここでもし、キャッシュメモリ内に該当するデータが存在している場合には、ＣＰＵにそれを渡す。一方、存在しない場合には、バス４を介してメインメモリ７上のデータを要求する。これにより、他のプロセッサモジュールのキャッシュメモリ、またはメインメモリ７からデータを受け取ることができるとともに、読み出しを行ったＣＰＵのキャッシュメモリにこのデータを保持することができる。

【００２２】バスコントローラ５は、プロセッサモジュール１ａ～１ｃとメインメモリ７間のデータ転送を司るものであり、メインメモリ７の更新履歴を保持するログメモリ６ａ、６ｂを制御する機能も有している。このため、バスコントローラ５は、ログメモリ６ａ、６ｂにそれぞれ対応したログカウンタ５２ａ，５２ｂを持つ。また、バスコントローラ５は、各プロセッサモジュール１ａ～１ｃがどのログメモリを使うべきかを決定するために使用されるログテーブル５１も有している。図２には、ログテーブル５１の構造が示されている。

【００２３】図２に示されているように、ログテーブル５１は、プロセッサモジュール１ａ～１ｃそれぞれのＣＰＵ番号とそれらＣＰＵが現在使用しているログメモリの番号（カレントログメモリ番号）との対応関係を保持している。ここで、カレントログメモリ番号“０”はログメモリ６ａを使用することを示し、カレントログメモリ番号“１”はログメモリ６ｂを使用することを示す。このシステムのスタートアップ時においては、全てのＣＰＵのカレントログメモリ番号は“０”を示しており、すべてのＣＰＵはログメモリ６ａを使うように設定される。

【００２４】図３には、２つのログカウンタ５２ａ，５２ｂと２つのログメモリ６ａ，６ｂとの関係が示されている。ログカウンタ５２ａは、ログメモリ６ａの更新履歴情報格納位置を指定するポインタを保持しており、そのポインタ値は更新履歴情報がログメモリ６ａに書き込まれる度に、ログメモリ６ａの先頭番地から最終番地に向けて＋１ずつインクリメントされる。ログカウンタ５２ｂは、ログメモリ６ｂの更新履歴情報格納位置を指定するポインタを保持しており、そのポインタ値は更新履歴情報がログメモリ６ｂに書き込まれる度に、ログメモ

リ６ｂの先頭番地から最終番地に向けて＋１ずつインクリメントされる。

【００２５】通常のプログラム実行処理の期間においては、各ＣＰＵは、メインメモリ７への書き込みが必要となった時に、バスコントローラ５に対し、

（１）ＣＰＵのＩＤ

（２）メモリのアドレス

（３）メモリのデータ

を渡す。バスコントローラ５は、この書き込み要求を検出すると、ＣＰＵのＩＤ（ＣＰＵ番号）から、どのログメモリを使うか決定し（ここではログメモリ６ａを使用するものとする）、そのログメモリ６ａに対応したログカウンタ５２ａの値を得て、そのカウンタに対応したログメモリ６ａの位置に、メモリアドレスとそのメモリアドレスの更新前データとを更新履歴情報として記録する。そして、メインメモリ７を更新する。

【００２６】その後、プロセッサモジュール１ａ～１ｃの各ＣＰＵは、所定の時間が経過したこと、あるいはログメモリ６ａの残容量が所定の量を下回ったことをバスコントローラ５からの割り込み信号やポーリング処理によって検知したとき、チェックポイント取得処理を開始する。チェックポイント取得処理では、各ＣＰＵの内部状態復元のために必要なレジスタの値、およびキャッシュメモリ内のデータのうち、まだメインメモリ７に反映されていないデータが、バスコントローラ５を介して、メインメモリ７に書き込まれる。この場合の更新履歴についても、前述と同様にログメモリ６ａに記録される。

【００２７】ここまで終了したプロセッサモジュール１ａ～１ｃの各ＣＰＵは、使用するログメモリを切り替えるためにログテーブル５１のカレントログメモリ番号を“０”から“１”に書き換え（ここではログメモリ６ｂを次に使用するものとする）、その後、即座に通常のプログラム処理を再開する。

【００２８】この様にチェックポイント取得処理を完了したＣＰＵの順で、使用するログメモリをそれまで使用していたログメモリから他方のログメモリに切り替えることにより、それまで使用していたログメモリの内容を破壊することなく、通常のプログラム実行処理を再開することができる。よって、各ＣＰＵが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようになり、システム全体の待機時間を大幅に減少させることができる。

【００２９】次に、図４を参照して、チェックポイント取得処理全体の流れにつて説明する。

【００３０】いま、図４に示すように、プロセッサモジュール１ａ～１ｃのＣＰＵ２ａ～２ｃが並列に稼働しているものと、すなわち、各ＣＰＵ２ａ～２ｃが、通常のプログラム実行処理をそれぞれ行なっているものとする（図４の（１））。そして、このときに使用されているログメモリはログメモリ６ａであるとする（図４の（２））。

【００３１】その後、各ＣＰＵ２ａ～２ｃの稼働中にログメモリ６ａの残容量が予め設定された所定量を下回ったときに（図４の（３））、ＣＰＵ２ａ～２ｃそれぞれは、その旨を検知して、チェックポイント処理を開始する（図４の（４））。

【００３２】このチェックポイント取得処理の開始および所要時間は、上述したように、その検知タイミングや、検知したときに処理中のプログラムの種類、キャッシュメモリの状態などにより異なる。

【００３３】しかし、各ＣＰＵ２ａ～２ｃそれぞれは自身のチェックポイント処理が終了した際に、使用するログメモリを切り替えて、即座に通常処理を再開する（図４の（５））。これにより、通常処理を再開したＣＰＵについては、ログメモリ６ｂへの更新履歴の採取が開始される（図４の（６））。

【００３４】一方、すべてのＣＰＵ２ａ～２ｃのチェックポイント処理が終了した際に（図４の（７））、ログメモリ６ａのリセット、すなわち、ログメモリ６ａに記録された更新履歴の破棄が行なわれる。

【００３５】次に、図５のフローチャートを参照して、各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順について説明する。

【００３６】各プロセッサモジュール１ａ～１ｃのＣＰＵは、通常処理を行なっているときに（ステップＳ１）、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合（ステップＳ２）、チェックポイント取得処理を開始する（ステップＳ３）。このチェックポイント取得処理では、ＣＰＵの各種レジスタを含むそのＣＰＵの内部状態（コンテキスト）と、メインメモリ７に反映されてないキャッシュメモリの内容がメインメモリ７に書き込まれる。次に、そのＣＰＵは、使用するログメモリを切り替えるためにログテーブル５１の該当するカレントログメモリ番号を“０”から“１”に、書き換える（ステップＳ４）。この後、そのＣＰＵは、自身がチェックポイント取得処理を完了した最後のＣＰＵであるか否かを判断する（ステップＳ５）。これは、ログテーブル５１を参照して、カレントログメモリ番号を書き換えてないＣＰＵが存在するかどうかを調べることによって行われる。全てのＣＰＵのカレントログメモリ番号が新たな番号に書き換えられていれば、そのＣＰＵがチェックポイント取得処理を完了した最後のＣＰＵである。この場合、そのＣＰＵは、今まで使用していたログメモリの内容を廃棄するために、そのログメモリをクリアする（ステップＳ６）。このクリア処理では、そのログメモリに対応するログカウンタのポインタ値が“０”に戻される。この後、そのＣＰＵは、チェックポイント取得処理で中断した通常のプログラム実行処理を再開する（ステップＳ７）。

【００３７】図１のマルチプロセッサシステムになんら

13

かの障害が発生した場合には、最後に使い始めたログメモリにスタック形式で保持されている更新履歴情報が逐次読み出され、更新前データがメインメモリ７上の対応する番地に書き込まれる。これにより、メインメモリ７が障害発生前のチェックポイントの時点に復元される。もし、もう一方のログメモリについても、それに対応するログカウンタのポインタ値が“０”でなければ、同様の処理が行われる。

【００３８】これにより、障害以前のチェックポイントを取った時点のメインメモリ７の内容が復元され、そのメモリ７にセーブされていたレジスタなどの内容を使ってＣＰＵの内部状態が復元できる。そして、この状態で障害の原因を解明し、それを除去してから実行を再開すれば、障害があった場合でも、システム全体に異常を起こさずに、処理を進めることが可能となる。

【００３９】（第２実施形態）次に、この発明の第２実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第１実施形態に比べ、各プロセッサのキャッシュメモリのライトポリシー（ライトスルー／ライトキャッシュ）を動的に切り替える機能を有する点だけが異なり、他の点は全て第１実施形態と同じである。

【００４０】ここで、キャッシュメモリと同時にメインメモリにもデータを書き込む方式をライトスルー（write through）という。この方式では、書き込みの時間はメインメモリ７のアクセス時間と同じなので、通常処理の高速化はあまり期待できない。しかし、チェックポイント取得処理でキャッシュの内容をフラッシュする必要がないので、チェックポイント取得処理に要する時間を短くすることができる。

【００４１】これに対してライトバック（write back）方式では、書き込みはキャッシュメモリにだけおこなわれ、キャッシュフラッシュするときに、キャッシュメモリの内容がメインメモリに書き戻される。したがって、ライトバック方式は、ライトスルー方式の場合に比べて、チェックポイント取得処理に要する時間は長くなるものの、通常処理においては、バストラフィックを低減でき、システム全体のパフォーマンスの向上を実現できる。

【００４２】したがって、このマルチプロセッサシステムでは、基本的には、全てのプロセッサモジュールのキャッシュメモリをライトバック方式で動作させておき、一定の時間内で処理を終了させることが必要なリアルタイム処理を実行する場合に、一部のプロセッサモジュールについてのみそのキャッシュメモリをライトスルー方式に切り替えて、そのプロセッサモジュールにリアルタイム処理を実行させるという制御が行われる。このようなキャッシュのライトポリシーの切り替え処理は、このマルチプロセッサシステム上で動作するプログラムの制御下で実行される。

14

【００４３】また、システムの稼働状態をモニタするプログラムを用いることにより、マルチプロセッサシステムのスループットおよびレスポンスタイムに基づいてライトスルー方式で動作するプロセッサモジュールの数を動的に変更することが好ましい。そのためのソフトウェア構成を図６に示す。

【００４４】図６は、図１のマルチプロセッサシステムを用いて構築したトランザクション処理システムを示す。このトランザクション処理システムは、複数のクライアントコンピュータ５０１と、図１のマルチプロセッサシステムから構成されるサーバコンピュータ６０１とを有している。サーバコンピュータ６０１においては、複数のクライアントコンピュータ５０１それぞれからのトランザクション処理要求が通信プロセッサ３００によって受け付けられる。それらトランザクション処理要求は、トランザクション管理プログラム２００によって各種アプリケーションプログラムに振り分けられて処理される。そして、その処理結果がトランザクション管理プログラム２００および通信プロセッサ３００を介してクライアントコンピュータ５０１それぞれに通知される。

【００４５】トランザクション管理プログラム２００は、このトランザクション処理システムの稼働状態をモニタしており、スループット（単位時間当たりのトランザクション処理要求数）とレスポンスタイム（トランザクション処理要求が受け付けられてからそれに対する処理結果が通知されるまでの応答時間）とに基づいて、ライトスルー方式で動作するプロセッサモジュールの数を動的に変更する機能を持つ。

【００４６】ここでは、図６に示されているように、図１のキャッシュメモリ３ａ～３ｃがそれぞれＣＰＵ２ａ～２ｃの一次キャッシュとして実現されている場合を例にとって、キャッシュのライトポリシー変更動作を原理を説明する。

【００４７】キャッシュメモリ３ａ～３ｃそれぞれのライトポリシー（ライトバック／ライトスルー）は、対応するＣＰＵ２ａ～２ｃ内部のライトポリシー制御レジスタＲ１～Ｒ３に設定される値によって決定される。このため、トランザクション管理プログラム２００は、それらライトポリシー制御レジスタＲ１～Ｒ３に書き込む値を変更することによって、ライトスルー方式で動作するキャッシュを持つＣＰＵ数を制御する。この場合、どのＣＰＵが現在ライトスルーキャッシュを持つＣＰＵとして動作しているかについては、トランザクション管理プログラム２００によって管理されている。

【００４８】次に、図７を参照して、ライトスルーキャッシュとライトバックキャッシュが混在する場合のチェックポイント取得処理全体の流れについて説明する。ここでは、図４と異なる部分についてのみ説明する。

【００４９】いま、ＣＰＵ２ａ，２ｂの備えるキャッシュメモリをライトバックで動作させ、ＣＰＵ２ｃの備え

15

るキャッシュメモリをライトスルーで動作させたとすると、ＣＰＵ２ｃのチェックポイント処理（図７の（４）′）に費やす時間は、ＣＰＵ２ａ～２ｂのチェックポイント処理（たとえば図７の（４））と比較して、大幅に短縮されることになり、常に所定の時間内に終了させる必要がある処理（リアルタイム処理）などをこのＣＰＵ２ｃで処理させるようにすれば、チェックポイント取得に伴う弊害を回避することが可能となる。これは、ライトスルーキャッシュを持つＣＰＵがチェックポイント取得処理で行なわなければならないことは、そのＣＰＵの内部状態を保存するだけとなり、キャッシュメモリの内容を書き出す処理が不要であるためである。

【００５０】このため、チェックポイントの動作が長い時間継続することにより、一定時間内に動作しなければならないようなリアルタイムプロセスに悪影響を与えるといったことがほとんどなくなり、レスポンスタイムを向上することができる。

【００５１】次に、図８のフローチャートを参照して、トランザクション管理プログラム２００によって実行される、システムの稼働状態に応じてライトスルーキャッシュを持つＣＰＵとして動作するＣＰＵ数を動的に変更する処理の手順について説明する。

【００５２】トランザクション管理プログラム２００は、まず、ライトスルーキャッシュを持つＣＰＵとして動作するＣＰＵ（プロセッサ）数を示す変数（ｎ）に値“０”を代入して、全てのＣＰＵ２ａ～２ｃをライトバックキャッシュを持つＣＰＵとして動作させる（ステップＳ１１）。この状態でトランザクション要求の受付および応答が開始される。トランザクション管理プログラム２００は、ある一定時間間隔、例えば５分単位でスループットＴＡおよび平均レスポンスタイムＲＡを求める（ステップＳ１２）。そして、変数（ｎ）の値を＋１し、ある１つのＣＰＵのキャッシュメモリのライトポリシーをライトバックからライトスルーに変更する（ステップＳ１３）。そして、その後の５分間についてのスループットＴＢおよび平均レスポンスタイムＲＢを求める（ステップＳ１４）。

【００５３】次いで、トランザクション管理プログラム２００は、スループットＴＡとスループットＴＢとの差分を求め、その差がある一定値αよりも少ないか否か、つまり変数（ｎ）を変更する前と後とでスループットがあまり変動してないかどうかを調べる（ステップＳ１５）。スループットの変動が少ないならば、トランザクション管理プログラム２００は、変数（ｎ）を変更する前よりも変更した後の方がレスポンスタイムが減少しているかどうかを調べるために、レスポンスタイムＲＡからレスポンスタイムＲＢの値を減算し、その差分が一定値βよりも大きいか否かを判断する（ステップＳ１６）。一定値βよりも小さいならば、トランザクション管理プログラム２００は、変数（ｎ）の値を－１して、

16

ライトスルーキャッシュを持つＣＰＵとして動作するＣＰＵの数を１つ減らす（ステップＳ１７）。一方、一定値βよりも大きいならば、ライトスルーキャッシュを持つＣＰＵとして動作するＣＰＵの数は変更しない。

【００５４】ステップＳ１２～ステップＳ１７の処理は、システムが稼働している間、繰り返し実行される。これにより、ライトスルー方式のキャッシュメモリとライトバック方式のキャッシュメモリとが適切な数で混在させることになる。

【００５５】（第３実施形態）次に、この発明の第３実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第１実施形態に比べ、２つのログメモリ６ａ，６ｂが１個の物理メモリ上に実現されている点だけが異なり、他の点は全て第１実施形態と同じである。

【００５６】図９には、２つのログメモリ６ａ，６ｂを１個の物理メモリ上に実現するための構成が示されている。

【００５７】ログメモリ６は、通常のランダムアクセスメモリであり、０番地からＭ－１番地までアドレスづけがされており、ひとつの番地にひとつの更新履歴が格納できるように構成されている。このログメモリ６の記憶空間上に２つのログメモリ６ａ，６ｂが論理的に実現される。

【００５８】バスコントローラ５には、前述したログテーブル５１、２つのログカウンタ５２ａ，５２ｂに加え、メモリ制御ロジック２０１、マルチプレクサ２０２、加算器２０３、減算器２０４が設けられている。

【００５９】ログカウンタ５２ａ，５２ｂは、それぞれ更新履歴情報を書き込むべきログメモリ６上のアドレスを保持するのに使われる。ログカウンタ５２ａに保持されるポインタ値はログメモリ６ａを実現するために使用され、ログメモリ６ａに更新履歴情報が書き込まれる度に、ログメモリ６の０番地からＭ－１番地に向かって加算器２０３によって＋１ずつ順次インクリメントされる。ログカウンタ５２ｂに保持されるポインタ値はログメモリ６ｂを実現するために使用され、ログメモリ６ｂに更新履歴情報が書き込まれる度に、ログメモリ６のＭ－１番地から０番地に向かって減算器２０４によって－１ずつ順次デクリメントされる。

【００６０】したがって、０番地からログカウンタ５２ａが指すアドレス－１までがログメモリ６ａに相当し、Ｍ－１番地からログカウンタ５２ｂが指すアドレス＋１までがログメモリ６ｂに相当する。メモリ制御ロジック２０１は、更新履歴情報（メインメモリのアドレス、更新前データ）をライトデータとしてログメモリ６に供給すると共に、書き込み要求を発行したＣＰＵ番号（プロセッサＩＤ）をマルチプレクサ２０２に送る。

【００６１】マルチプレクサ２０２は、受け取ったＣＰＵ番号に対応するカレントログメモリ番号をログテーブ

ル５１から得、そのカレントログメモリ番号に対応するログカウンタのポインタ値を選択してそれをライトアドレスとしてログメモリ６に送る。

【００６２】以下、図１０を参照して、図９のバスコントローラ５の動作を説明する。

【００６３】初期状態では、ログカウンタ５２ａのポインタ値は０、ログカウンタ５２ｂのポインタ値はＭ－１となっている。これは、ログメモリ６ａとログメモリ６ｂのどちらにも更新履歴情報が格納されていないことを意味する。

【００６４】ＣＰＵ２ａ～２ｃのいずれかからメインメモリ７に対する書き込み要求（更新トランザクション）をバスコントローラ５が受け取ると（ステップＳ２１）、メモリ制御ロジック２０１は、メインメモリ７から更新前データを読み込み、その更新前データとメインメモリ７のメモリアドレスとを含む更新履歴情報をログメモリ６に送る（ステップＳ２２）。この後、マルチプレクサ２０２は、書き込み要求を発行したＣＰＵ番号に対応するカレントログメモリ番号をログテーブル５１から読み取り（ステップＳ２３）、ログカウンタ５２ａ，５２ｂの中で、読み取ったカレントログメモリ番号に対応するログカウンタのポインタ値を選択する（ステップＳ２４）。これにより、その選択されたログカウンタのポインタ値が指定するログメモリ６の番地に更新履歴情報が書き込まれる。この後、その選択されたログカウンタのポインタ値が更新される（ステップＳ２５）。

【００６５】例えば、書き込み要求を発行したＣＰＵ番号に対応するカレントログメモリ番号がログメモリ６ａを指定している場合には、ログカウンタ５２ａのポインタ値が選択され、そのポインタ値が示す番地に更新履歴情報の書き込みが行われる。そして、ログカウンタ５２ａから読み出されたポインタ値は、加算器２０３で＋１されて、再びログカウンタ５２ａに格納される。同様に、書き込み要求を発行したＣＰＵ番号に対応するカレントログメモリ番号がログメモリ６ｂを指定している場合には、ログカウンタ５２ｂのポインタ値が選択され、そのポインタ値が示す番地に更新履歴情報の書き込みが行われる。そして、ログカウンタ５２ｂから読み出されたポインタ値は、減算器２０４で－１されて、再びログカウンタ５２ｂに格納される。

【００６６】次に、図１１のフローチャートを参照して、この第３実施形態において各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順について説明する。

【００６７】各プロセッサモジュール１ａ～１ｃのＣＰＵは、通常処理を行なっているときに（ステップＳ３１）、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合（ステップＳ３２）、チェックポイント取得処理を開始する（ステップＳ３３）。ログメモリの残容量は、ログカウンタ５２ａに保持されているポインタ値とログカウンタ５２ｂに保持されているポインタ値の差として求めることができる。チェックポイント取得処理では、ＣＰＵの各種レジスタを含むそのＣＰＵの内部状態（コンテキスト）と、メインメモリ７に反映されてないキャッシュメモリの内容がメインメモリ７に書き込まれる。ライトスルーキャッシュの場合には、その内容がすでにメインメモリ７に反映済みであるので、ＣＰＵの内部状態だけがメインメモリ７に書き込まれる。

【００６８】次に、そのＣＰＵは、使用するログメモリを切り替えるためにログテーブル５１の該当するカレントログメモリ番号を“０”から“１”に、書き換える（ステップＳ３４）。この後、そのＣＰＵは、自身がチェックポイント取得処理を完了した最後のＣＰＵであるか否かを判断する（ステップＳ３５）。これは、ログテーブル５１を参照して、カレントログメモリ番号を書き換えてないＣＰＵが存在するかどうかを調べることによって行われる。全てのＣＰＵのカレントログメモリ番号が新たな番号に書き換えられていれば、そのＣＰＵがチェックポイント取得処理を完了した最後のＣＰＵである。この場合、そのＣＰＵは、今まで使用していたログメモリの内容を廃棄するために、そのログメモリをクリアする（ステップＳ３６）。このクリア処理では、そのログメモリがログメモリ６ａであればログカウンタ５２ａのポインタ値が“０”に戻され、またログメモリがログメモリ６ｂであればログカウンタ５２ｂのポインタ値が“Ｍ－１”に戻される。この後、そのＣＰＵは、チェックポイント取得処理で中断した通常のプログラム実行処理を再開する（ステップＳ３７）。

【００６９】マルチプロセッサシステムになんらかの障害が発生した場合には、最後に使い始めたログメモリにスタック形式で保持されている更新履歴情報が逐次読み出され、更新前データがメインメモリ７上の対応する番地に書き込まれる。これにより、メインメモリ７が障害発生前のチェックポイントの時点に復元される。もし、もう一方のログメモリについても、それに対応するログカウンタのポインタ値が初期値（ログカウンタ５２ａの場合は“０”、ログカウンタ５２ｂの場合は“Ｍ－１”）でなければ、同様の処理が行われる。

【００７０】図１２には、この第３実施形態のシステムの各プロセッサモジュールが通常動作とチェックポイント処理を繰り返し行なった場合におけるログカウンタ５２ａ，５２ｂのポインタ値の変化の様子が示されている。

【００７１】Ｔ０は初期状態であり、ログカウンタ５２ａのポインタ値は０、ログカウンタ５２ｂのポインタ値はＭ－１である。この時、ログメモリ６ａ，６ｂには履歴情報は格納されていない。また、全てのＣＰＵのカレントログメモリ番号はログメモリ６ａを示す。この状態から通常のプログラム実行処理が開始される。

19

【００７２】Ｔ１の時点では、ログメモリ６ａの残容量、すなわちログカウンタ５２ｂに保持されたポインタ値とログカウンタ５２ａに保持されたポインタ値の差が、予め設定された所定量を下回る。それを検出した各ＣＰＵは、順次チェックポイント取得処理を開始する。

【００７３】Ｔ２の時点では、最初に自身のチェックポイント取得処理を終了したＣＰＵが、ログメモリ６ａからログメモリ６ｂへの切り替えを行ない、即座に通常のプログラム実行処理に戻る。以降、このＣＰＵの通常処理においては、ログメモリ６ｂが更新履歴情報の格納に利用される。

【００７４】Ｔ３の時点では、最後に自身のチェックポイント処理を終了したＣＰＵが、ログメモリ６ａからログメモリ６ｂへの切り替えと、ログメモリ６ａの初期化を行ない、通常のプログラム実行処理に戻る。ここで、ログメモリ６ａの初期化、すなわちログメモリ６ａの更新履歴情報の破棄は、ログカウンタ５２ａのポインタ値を０に戻すことによって行われる。

【００７５】Ｔ４の時点では、ログメモリ６ｂの残容量、すなわちログカウンタ５２ｂに保持されたポインタ値とログカウンタ５２ａに保持されたポインタ値の差が、予め設定された所定量を下回る。それを検出した各ＣＰＵが、順次チェックポイント取得処理を開始する。

【００７６】Ｔ５の時点では、最初に自身のチェックポイント処理を終了したＣＰＵが、ログメモリ６ｂからログメモリ６ａへの切り替えを行ない、即座に通常のプログラム実行処理に戻る。この通常処理においては、ログメモリ６ａが更新履歴情報の格納に利用される。

【００７７】Ｔ６の時点では、最後に自身のチェックポイント処理を終了したＣＰＵが、ログメモリ６ｂからログメモリ６ａへの切り替えと、ログメモリ６ｂの初期化を行ない、通常のプログラム実行処理に戻る。ここで、ログメモリ６ｂの初期化、すなわちログメモリ６ｂの更新履歴情報の破棄は、ログカウンタ５２ｂのポインタ値をＭ－１に戻すことによって行われる。

【００７８】Ｔ０～Ｔ２では、すべてのＣＰＵが更新履歴をログメモリ６ａに格納するため、ログカウンタ５２ａのポインタ値が増加し、ログカウンタ５２ｂのポインタ値はＭ－１で一定である。

【００７９】Ｔ２～Ｔ３では、一部のＣＰＵが更新履歴をログメモリ６ａに格納し、残りのＣＰＵが更新履歴をログメモリ６ｂに格納するため、ログカウンタ５２ａのポインタ値が増加し、ログカウンタ５２ｂのポインタ値は減少する。

【００８０】Ｔ３の時点で、ログメモリ６ａの初期化により、ログカウンタ５２ａのポインタ値は０となる。

【００８１】Ｔ３～Ｔ５では、すべてのＣＰＵが更新履歴をログメモリ６ｂに格納するため、ログカウンタ５２ｂのポインタ値が減少し、ログカウンタ５２ａのポインタ値は０で一定である。

20

【００８２】Ｔ５～Ｔ６では、一部のＣＰＵが更新履歴をログメモリ６ａに格納し、残りのＣＰＵが更新履歴をログメモリ６ｂに格納するため、ログカウンタ５２ａのポインタ値が増加し、ログカウンタ５２ｂのポインタ値は減少する。

【００８３】Ｔ６の時点で、ログメモリ６ｂの初期化により、ログカウンタ５２ｂのポインタ値はＭ－１となる。

【００８４】以上のように、２つのログカウンタ５２ａ，５２ｂのポインタ値の一方をログメモリ６の先頭番地から最終番地に向かって順次インクリメントし、他方を最終番地から先頭番地に向かって順次デクリメントすることにより、物理的に１つのログメモリ６を使って２つのログメモリ６ａ，６ｂを実現できる。これは、単純にログメモリを２個設けた場合と比較して、ハードウェア量をほぼ半減させることができる。

【００８５】（第４実施形態）次に、この発明の第４実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第１実施形態に比べ、２つのログメモリ６ａ，６ｂが１個の物理メモリ上に実現されている点だけが異なり、他の点は全て第１実施形態と同じである。

【００８６】図１３には、２つのログメモリ６ａ，６ｂを１個の物理メモリ上に実現するための構成が示されている。

【００８７】ログメモリ６は、通常のランダムアクセスメモリであり、０番地からＭ－１番地までアドレスづけがされており、ひとつの番地で指定されるエントリに更新履歴情報とログメモリ番号とを対応づけて格納できるように構成されている。すなわち、各エントリは、メモリアドレス格納フィールド、更新前データ格納フィールド、ログメモリ番号格納フィールドを含む。

【００８８】バスコントローラ５には、前述したログテーブル５１に加え、メモリ制御ロジック３０１、アドレス制御ロジック３０２が設けられている。また、ログカウンタ５２ａ，５２ｂの代わりに、３つのログカウンタ３０３～３０５と、加算器３０６が設けられている。

【００８９】ログカウンタ３０３は、次に更新履歴情報を書き込むべき番地を示すポインタ値を保持するのに使われる。このログカウンタ３０３のポインタ値は、書き込み要求を発行したＣＰＵがログメモリ６ａ，６ｂのどちらをカレントログメモリとして指定している場合でも、更新履歴情報を書き込む度に、加算器３０６によって番地０から順に＋１ずつインクリメントされる。そして、そのポインタ値がログメモリ６の最後の番地すなわちＭ－１に達すると、次は０に戻ってそこから再び＋１ずつインクリメントされる。したがって、ログカウンタ３０３のポインタ値は、０，１，～，Ｍ－１，０，１，～，Ｍ－１，０，……を繰り返すことになる。

【００９０】ログカウンタ３０４は、現在使用中のログ

メモリに対応する履歴情報の中で最初のものが格納されている番地をログカウンタ３０５に素早く設定するためにその番地を一時的に保持する。ログカウンタ３０５は、現在使用中のログメモリに対応する履歴情報のうちで最初のものが格納されている番地を示すポインタを保持するのに使われる。例えば、すべてのＣＰＵがあるチェックポイント処理を終了し、それまで使われてきたログメモリ６ａに格納されている更新履歴情報が不要となった際には、次のログメモリ６ｂのログメモリ番号が付されている更新履歴情報の最初のものを指すように、ログカウンタ３０４に保持されているポインタ値がログカウンタ３０５に転送され、ログカウンタ３０５に保持されているポインタ値の更新が行なわれる。これらログカウンタ３０４，３０５は、それぞれレジスタによって構成することができる。

【００９１】メモリ制御ロジック３０１は、書き込み要求を発行したＣＰＵ番号に対応するログメモリ番号をログテーブル５１から読み取り、その読み取ったログメモリ番号と更新履歴情報とをライトデータとしてログメモリ６に供給する。アドレス制御ロジック３０２は、ログカウンタ３０３～３０５の制御、およびログメモリ６に与えるアドレスを制御するものであり、メモリ制御ロジック３０１から更新履歴情報とログメモリ番号を受け取ると、ログカウンタ３０３に格納されているポインタ値を読み出し、そのポインタ値をログメモリ６にライトアドレスとして与える。また、アドレス制御ロジック３０２は、チェックポイント取得処理を最初に完了したＣＰＵから最初の書き込み要求が発行された時、つまりメモリ制御ロジック３０１から新しいログメモリ番号を受け取った時に、その時のログカウンタ３０３のポインタ値をログカウンタ３０４に転送する。さらに、アドレス制御ロジック３０２は、ログテーブル５１を参照して全てのＣＰＵのチェックポイント取得処理が完了したことを検知したとき、その時のログカウンタ３０３のポインタ値をログカウンタ３０４に転送する。

【００９２】なお、これらログカウンタ３０３～３０５は各ＣＰＵによってリード・ライト可能なＩ／Ｏレジスタによって実現できるので、ログカウンタ間のポインタ値の転送はＣＰＵの制御の下で行うことも可能である。

【００９３】以下、図１４を参照して、図１３のバスコントローラ５の動作を説明する。

【００９４】初期状態では、ログカウンタ３０３～３０５のポインタ値はいずれも０である。これは、ログメモリ６ａとログメモリ６ｂのどちらにも更新履歴情報が格納されていないことを意味する。

【００９５】ＣＰＵ２ａ～２ｃのいずれかからメインメモリ７に対する書き込み要求（更新トランザクション）をバスコントローラ５が受け取ると（ステップＳ４１）、メモリ制御ロジック３０１は、メインメモリ７から更新前データを読み込む（ステップＳ４２）。次いで、メモリ制御ロジック３０１は、書き込みを要求したＣＰＵに対応するログメモリ番号をログテーブル５１から得え、メモリアドレスおよび更新前データを含む更新履歴情報にログメモリ番号を付加してログメモリ６に送る（ステップＳ４３）。この後、アドレス制御ロジック３０２は、ログカウンタ３０３のポインタ値をライトアドレスとしてログメモリ６２与え、これによってそのポインタ値が指定するログメモリ６の番地に更新履歴情報とログメモリ番号が書き込まれる（ステップＳ４４）。この後、ログカウンタ３０３のポインタ値が加算器３０６によって＋１インクリメントされる（ステップＳ４５）。

【００９６】次に、図１５のフローチャートを参照して、この第４実施形態において各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順について説明する。

【００９７】各プロセッサモジュール１ａ～１ｃのＣＰＵは、通常処理を行なっているときに（ステップＳ５１）、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合（ステップＳ５２）、チェックポイント取得処理を開始する（ステップＳ５３）。ここで、ログメモリの残容量は、次のように求められる。

【００９８】ログカウンタ３０３のポインタ値をＰ１、ログカウンタ３０５のポインタ値をＰ２、ログメモリ６のサイズをＭとすると、Ｐ１＞Ｐ２であれば、残り容量は、Ｍ－（Ｐ１－Ｐ２）で与えられ、Ｐ１＞Ｐ２でなければ、残り容量は、Ｐ２－Ｐ１で与えられる。

【００９９】Ｐ１＞Ｐ２の場合におけるログメモリ６の利用状態とポインタＰ１，Ｐ２の関係を図１６（Ａ）に示す。

【０１００】ログメモリ６においてログメモリ番号０が格納されているエントリがログメモリ６ａとして使用される記憶領域であり、ログメモリ番号１が格納されているエントリがログメモリ６ｂとして使用される記憶領域である。ここでは、現在使用中のログメモリ６ｂに最初に更新履歴情報が格納された番地は３番地（ポインタ値Ｐ２）であり、次に更新履歴情報を格納すべき番地は１１番地（ポインタ値Ｐ１）ある。もし、この状態で障害が発生した場合には、ポインタ値Ｐ１を現在の番地から３番地まで戻しながら更新履歴情報を順次取り出し、ログメモリ６ｂを示すログメモリ番号が付加された更新履歴情報をメインメモリ７に書き込む処理が行われる。よって、現在有効に使用されているログメモリ６ｂのメモリサイズはＰ１－Ｐ２となるので、残り容量はＭ－（Ｐ１－Ｐ２）となる。

【０１０１】Ｐ２＞Ｐ１の場合におけるログメモリ６の利用状態とポインタＰ１，Ｐ２の関係を図１６（Ｂ）に示す。

【０１０２】ここでは、現在使用中のログメモリ６ｂに

23

最初に更新履歴情報が格納された番地は７番地（ポインタ値Ｐ２）であり、次に更新履歴情報を格納すべき番地は３番地（ポインタ値Ｐ１）ある。すなわち、ポインタ値Ｐ１は、Ｍ－１番地を越え、再び０番地から現在の３番地まで増加されたことになる。もし、この状態で障害が発生した場合には、ポインタ値Ｐ１を、２番地、１番地，０番地，Ｍ－１番地，……，７番地といった順序で戻しながら更新履歴情報を順次取り出し、ログメモリ６ｂを示すログメモリ番号が付加された更新履歴情報をメインメモリ７に書き込む処理が行われる。よって、ログメモリ６ｂの残りメモリサイズは、Ｐ２－Ｐ１となる。

【０１０３】図１５のステップＳ５３のチェックポイント取得処理では、ＣＰＵの各種レジスタを含むそのＣＰＵの内部状態（コンテキスト）と、メインメモリ７に反映されてないキャッシュメモリの内容がメインメモリ７に書き込まれる。ライトスルーキャッシュの場合には、その内容がすでにメインメモリ７に反映済みであるので、ＣＰＵの内部状態だけがメインメモリ７に書き込まれる。

【０１０４】次に、そのＣＰＵは、使用するログメモリを切り替えるためにログテーブル５１の該当するカレントログメモリ番号を“０”から“１”に、書き換える（ステップＳ５４）。この後、そのＣＰＵは、自身がチェックポイント取得処理を完了した最初のＣＰＵであるか否かを判断する（ステップＳ５５）。これは、ログテーブル５１を参照して、カレントログメモリ番号を書き換えたＣＰＵが他に存在するかどうかを調べることによって行われる。チェックポイント取得処理を完了した最初のＣＰＵであれば、そのＣＰＵは、ログカウンタ３０３のポインタ値をログカウンタ３０４にコピーする（ステップＳ５６）。

【０１０５】一方、チェックポイント取得処理を完了した最初のＣＰＵではない場合には、そのＣＰＵは、自身がチェックポイント取得処理を完了した最後のＣＰＵであるか否かを判断する（ステップＳ５７）。これは、ログテーブル５１を参照して、カレントログメモリ番号を書き換えてないＣＰＵが存在するかどうかを調べることによって行われる。全てのＣＰＵのカレントログメモリ番号が新たな番号に書き換えられていれば、そのＣＰＵがチェックポイント取得処理を完了した最後のＣＰＵである。この場合、そのＣＰＵは、ログカウンタ３０４のポインタ値をログカウンタ３０５にコピーする（ステップＳ５８）。

【０１０６】図１７には、この第４実施形態のシステムの各プロセッサモジュールが通常動作とチェックポイント処理を繰り返し行なった場合におけるログカウンタ３０３～３０５それぞれのポインタ値の変化の様子が示されている。

【０１０７】Ｔ０は初期状態で、ログカウンタ３０３～３０５はいずれも０を保持しており、ログメモリ６ａ，

24

６ｂには履歴情報が格納されていない状態である。また、全てのＣＰＵのカレントログメモリ番号はログメモリ６ａを示す。この状態から通常のプログラム実行処理が開始される。

【０１０８】Ｔ１の時点では、Ｐ１＞Ｐ２であるので、ログメモリ６ａの残容量（ログメモリ６の残り容量と同じ）は、Ｍ－（Ｐ１－Ｐ２）で与えられる。この残容量の値が予め設定された所定量を下回っている場合、それを検出した各ＣＰＵが、順次チェックポイント取得処理を開始する。

【０１０９】Ｔ２の時点では、最初に自身のチェックポイント取得処理を終了したＣＰＵが、ログメモリ６ａからログメモリ６ｂへの切り替えと、ログカウンタ３０３からログカウンタ３０４へのポインタ値のコピーを行ない、即座に通常のプログラム実行処理に戻る。以降、このＣＰＵの通常処理においては、ログメモリ６ｂが更新履歴情報の格納に利用される。

【０１１０】Ｔ３の時点では、最後に自身のチェックポイント取得処理を終了したＣＰＵが、ログメモリ６ａからログメモリ６ｂへの切り替えと、ログカウンタ３０４からログカウンタ３０５へのポインタ値のコピーを行ない、即座に通常のプログラム実行処理に戻る。ログカウンタ３０４からログカウンタ３０５へのポインタ値のコピーは、ログメモリ６ａの履歴情報の初期化を意味する。

【０１１１】Ｔ５の時点では、Ｐ２＞Ｐ１であるので、ログメモリ６ｂの残容量は、Ｐ２－Ｐ１で与えられる。その残り容量が予め設定された所定量を下回っている場合、それを検出した各ＣＰＵが、順次チェックポイント取得処理を開始する。

【０１１２】Ｔ６の時点では、最後に自身のチェックポイント処理を終了したＣＰＵが、ログメモリ６ｂからログメモリ６ａへの切り替えと、ログカウンタ３０３からログカウンタ３０４へのポインタ値のコピーを行ない、即座に通常のプログラム実行処理に戻る。以降、このＣＰＵの通常処理においては、ログメモリ６ａが更新履歴情報の格納に利用される。

【０１１３】Ｔ７の時点では、最後に自身のチェックポイント処理を終了したＣＰＵが、ログメモリ６ｂからログメモリ６ａへの切り替えと、ログカウンタ３０４からログカウンタ３０５へのポインタ値のコピーを行ない、即座に通常のプログラム実行処理に戻る。ログカウンタ３０４からログカウンタ３０５へのポインタ値のコピーは、ログメモリ６ｂの履歴情報の初期化を意味する。

【０１１４】Ｔ０～Ｔ２では、すべてのＣＰＵが更新履歴をログメモリ６ａに格納するため、ログカウンタ３０３に保持されているポインタ値が増加し、ログカウンタ３０４，３０５に保持されているポインタ値は０で一定である。格納される更新履歴には、すべてログメモリ６ａを示すログメモリ番号０が付加されている。

25

【０１１５】Ｔ２の時点で、ログカウンタ３０３に保持されているポインタ値が、ログカウンタ３０４に複写される。ログカウンタ３０５に保持されているポインタ値は変化しない。

【０１１６】Ｔ２～Ｔ３では、一部のＣＰＵが更新履歴をログメモリ６ａに格納し、残りのＣＰＵが更新履歴をログメモリ６ｂに格納するため、格納される更新履歴には、ログメモリ６ａのログメモリ番号とログメモリ６ｂのログメモリ番号とが混在している。ログカウンタ３０４，３０５に保持されているポインタ値は変わらない。

【０１１７】Ｔ３の時点で、ログメモリ６ａの初期化により、ログカウンタ３０４に保持されているポインタ値がログカウンタ３０５に複写される。これにより、ログカウンタ３０５のポインタ値は、ログメモリ６ｂのログメモリ番号を付された最初の更新履歴の格納エントリを指すようになる。

【０１１８】Ｔ４の時点で、ログカウンタ３０３に保持されているポインタ値がＭ－１に達し、次の履歴情報の書き込みにより、０に戻る。

【０１１９】Ｔ３～Ｔ６では、すべてのＣＰＵが更新履歴をログメモリ６ｂに格納するため、ログカウンタ３０３に保持されているポインタ値が増加し、ログカウンタ３０４，３０５に保持されているポインタ値は変化しない。格納される更新履歴には、すべてログメモリ６ｂのログメモリ番号が付されている。

【０１２０】Ｔ６の時点で、ログカウンタ３０３に保持されているポインタ値が、ログカウンタ３０４に複写される。ログカウンタ３０５に保持されているポインタ値は変化しない。

【０１２１】Ｔ７の時点で、ログメモリ６ｂの初期化により、ログカウンタ３０４に保持されているポインタ値がログカウンタ３０５に複写される。これにより、ログカウンタ３０５のポインタ値は、ログメモリ６ａのログメモの番号を付された最初の更新履歴の格納エントリを指すようになる。

【０１２２】以上示したように、第４実施形態ににおいても、単純にログメモリを２個設けた場合と比較して、ハードウェア量をほぼ半減させることができ、従来のひとつのログメモリとさほど変わらないハードウェアで実現することが可能である。

【０１２３】なお、第３および第４実施形態において、ログメモリ６は、通常のランダムアクセスメモリで実現できる。このことは、ログメモリ６は、メインメモリ７の一部を使って実現できることを意味する。

【０１２４】一般に、マルチプロセッサシステムにおいては、それがどのような処理を実行するサーバとして使用されるか、あるいは搭載されているＣＰＵの個数やＣＰＵ性能によって、メインメモリ７の更新の頻度や最適なチェックポイント取得の間隔が異なる。

【０１２５】従って、ログメモリ６をメインメモリ７の

26

一部を使って実現することにより、マルチプロセッサシステムが適用されるサーバの種類、ＣＰＵの個数やＣＰＵ性能、最適に設定されたチェックポイント間隔などに応じて、ログメモリ６の容量を最適に設定することが可能になる。

【０１２６】

【発明の効果】以上説明したように、本発明のマルチプロセッサシステムによれば、チェックポイント取得時におけるシステム全体の待機状態を大幅に減少させることが可能となり、かつ全体のパフォーマンスを低下させずに、リアルタイム処理を効率良く処理することが可能となる。また、複数のログメモリを従来のひとつのログメモリとさほど変わらないハードウェアで実現することが可能である。

【図面の簡単な説明】

【図１】この発明の第１実施形態に係るマルチプロセッサシステムの構成を示すブロック図。

【図２】図１のマルチプロセッサシステムに設けられたログテーブルの構成例を示す図。

【図３】図１のマルチプロセッサシステムに設けられた２つのログメモリと２つのログカウンタとの対応関係を示す図。

【図４】図１のマルチプロセッサシステムで実行されるチェックポイント取得処理全体の流れを示す図。

【図５】図１のマルチプロセッサシステムに設けられた各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順を示すフローチャート。

【図６】この発明の第２実施形態に係るマルチプロセッサシステムにおけるキャッシュライトポリシー制御の原理を説明するための図。

【図７】第２実施形態に係るマルチプロセッサシステムにライトスルーキャッシュとライトバックキャッシュが混在する場合のチェックポイント取得処理全体の流れを示す図。

【図８】第２実施形態に係るマルチプロセッサシステムにおいてライトスルーキャッシュを持つＣＰＵ数を変更する処理を示すフローチャート。

【図９】この発明の第３実施形態に係るマルチプロセッサシステムに設けられたバスコントローラとログメモリの構成を示す図。

【図１０】図９のバスコントローラの動作を示すフローチャート。

【図１１】第３実施形態のマルチプロセッサシステムに設けられた各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順を示すフローチャート。

【図１２】第３実施形態のマルチプロセッサシステムに設けられた２つのログカウンタのポインタ値の変化の様子を示す図。

【図１３】この発明の第４実施形態に係るマルチプロセ

27

ッサシステムに設けられたバスコントローラとログメモリの構成を示す図。

【図１４】図１３のバスコントローラの動作を示すフローチャート。

【図１５】第４実施形態のマルチプロセッサシステムに設けられた各プロセッサモジュールのＣＰＵが実行するチェックポイント取得処理の手順を示すフローチャート。

【図１６】第４実施形態のマルチプロセッサシステムにおけるログメモリの利用状態とその時の残り容量との関係を示す図。

【図１７】第４実施形態のマルチプロセッサシステムに

28

設けられた２つのログカウンタのポインタ値の変化の様子を示す図。

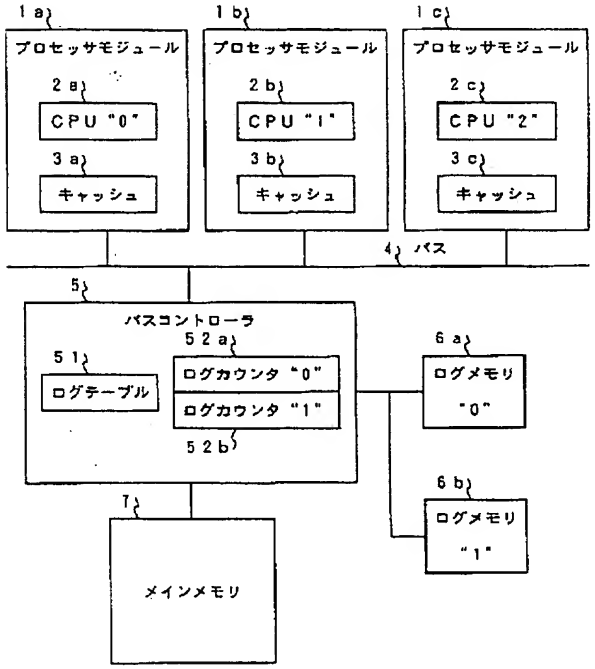
【図１８】ログメモリを持つ従来の典型的な持つ計算機システムの構成を示すブロック図。

【図１９】図１８の計算機システムで実行されるチェックポイント取得処理を説明するための図。

【符号の説明】

１ａ～１ｃ…プロセッサモジュール、２ａ～２ｃ…ＣＰＵ、３ａ～３ｃ…キャッシュメモリ、５…バスコントローラ、６ａ，６ｂ…ログメモリ、５１…ログテーブル、５２ａ，５２ｂ…ログカウンタ、２００…トランザクション管理プログラム、３００…通信プロセッサ。

【図１】

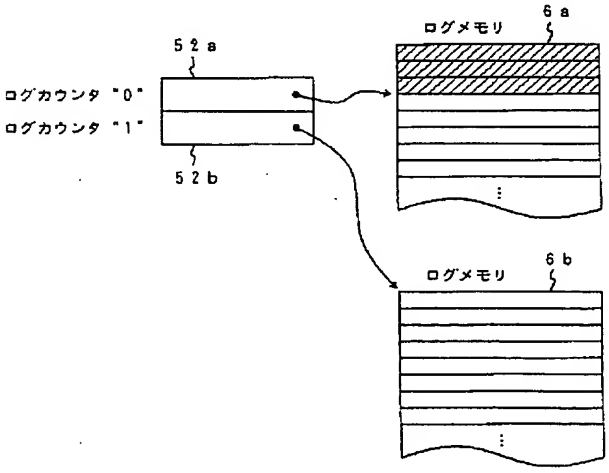


【図２】

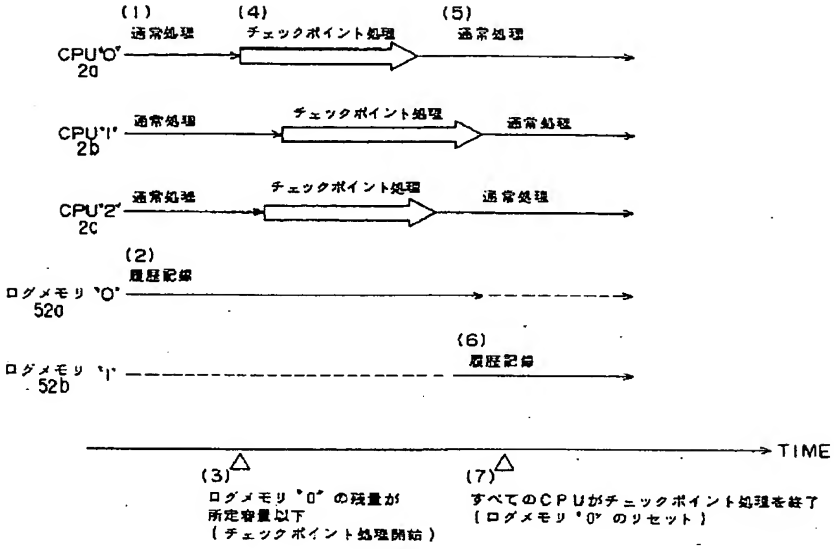
ログテーブル　５１

ＣＰＵ番号	カレントログメモリ番号
0	0／1
1	0／1
2	0／1
⋮	⋮

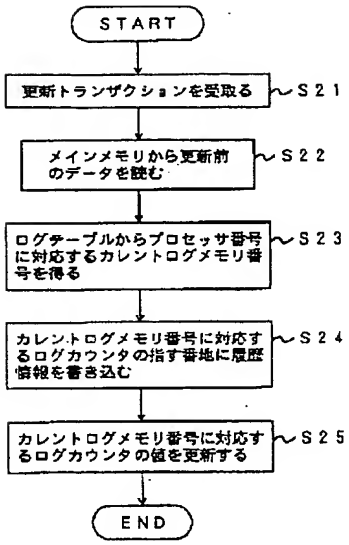
【図３】



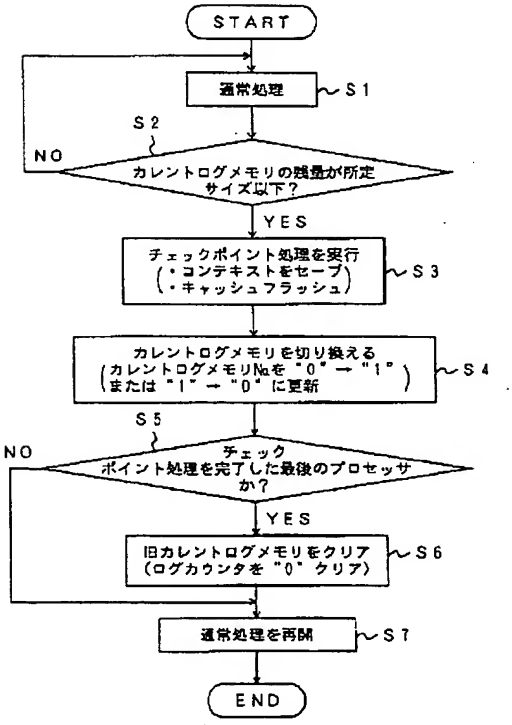
【図４】


(1)
通常処理
(4)
チェックポイント処理
(5)
通常処理
CPU"0"
20
CPU"1"
20
CPU"2"
20
(2)
履歴記録
ログメモリ "0"
52a
(6)
履歴記録
ログメモリ "1"
52b
TIME
(3)
ログメモリ "0" の残量が
所定容量以下
(チェックポイント処理開始)
(7)
すべてのCPUがチェックポイント処理を終了
(ログメモリ "0" のリセット)

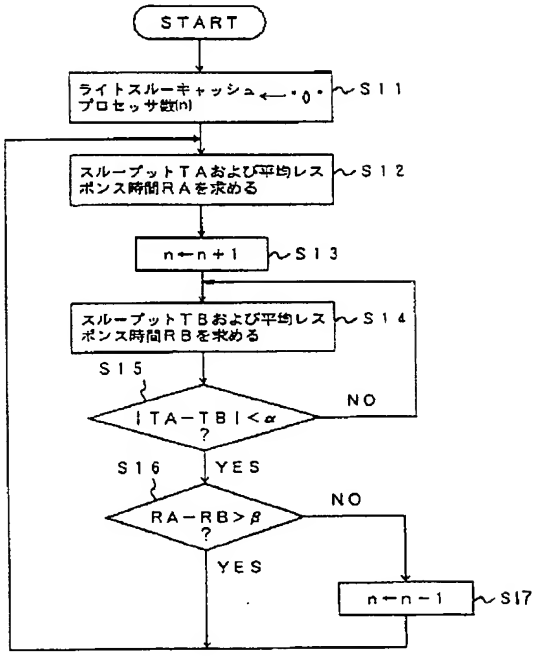
【図１０】


START
更新トランザクションを受取る
S21
メインメモリから更新前のデータを読む
S22
ログテーブルからプロセッサ番号に対応するカレントログメモリ番号を得る
S23
カレントログメモリ番号に対応するログカウンタの指す番地に履歴情報を書き込む
S24
カレントログメモリ番号に対応するログカウンタの値を更新する
S25
END

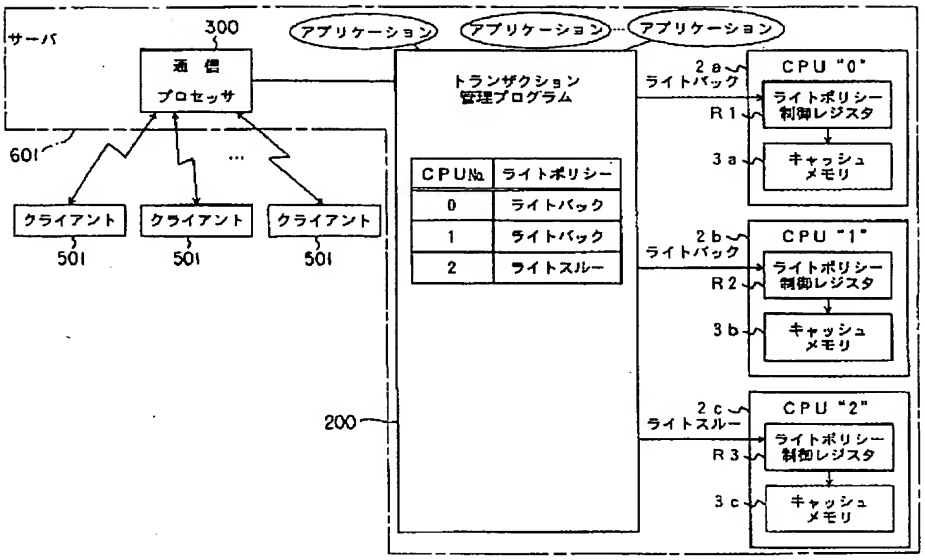
【図５】


START
通常処理
S1
S2
NO
カレントログメモリの残量が所定サイズ以下?
YES
チェックポイント処理を実行
(・コンテキストをセーブ)
(・キャッシュフラッシュ)
S3
カレントログメモリを切り換える
(カレントログメモリNoを "0" → "1"
または "1" → "0" に更新)
S4
S5
NO
チェックポイント処理を完了した最後のプロセッサか?
YES
旧カレントログメモリをクリア
(ログカウンタを "0" クリア)
S6
通常処理を再開
S7
END

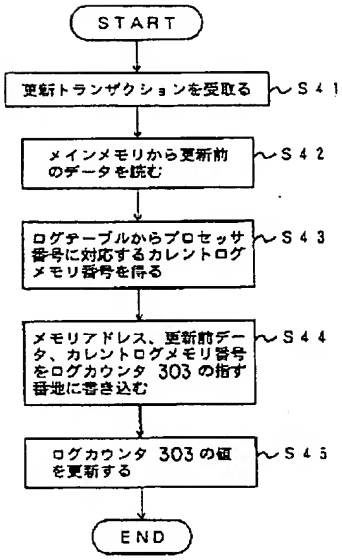
【図８】


START
ライトスルーキャッシュ ← "0"
プロセッサ数(n)
S11
スループットTAおよび平均レスポンス時間RAを求める
S12
n←n+1
S13
スループットTBおよび平均レスポンス時間RBを求める
S14
S15
|TA−TB|<α ?
NO
YES
S16
RA−RB>β ?
NO
YES
n←n−1
S17

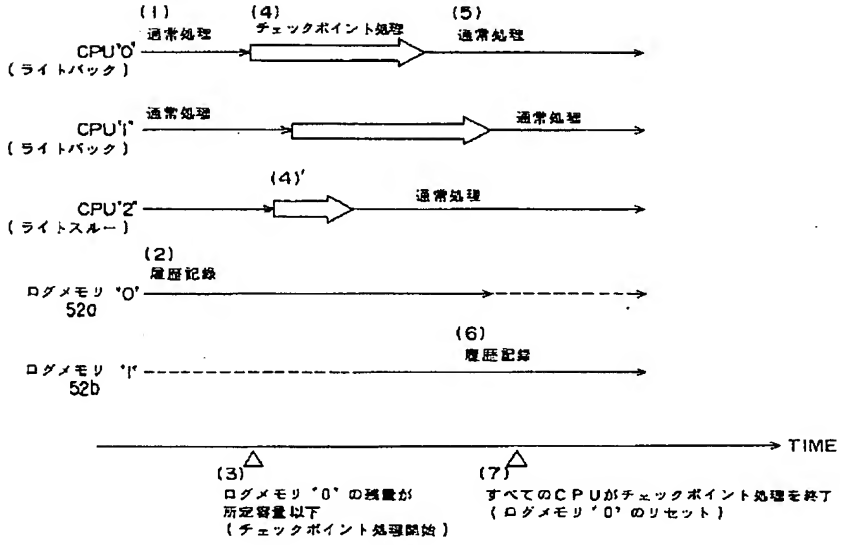
【図６】


サーバ
300
通信プロセッサ
601
クライアント
501
クライアント
501
クライアント
501
アプリケーション
アプリケーション
アプリケーション
トランザクション管理プログラム
CPU№
ライトポリシー
0
ライトバック
1
ライトバック
2
ライトスルー
200
2a
ライトバック
CPU "0"
ライトポリシー制御レジスタ
R1
3a
キャッシュメモリ
2b
ライトバック
CPU "1"
ライトポリシー制御レジスタ
R2
3b
キャッシュメモリ
2c
ライトスルー
CPU "2"
ライトポリシー制御レジスタ
R3
3c
キャッシュメモリ

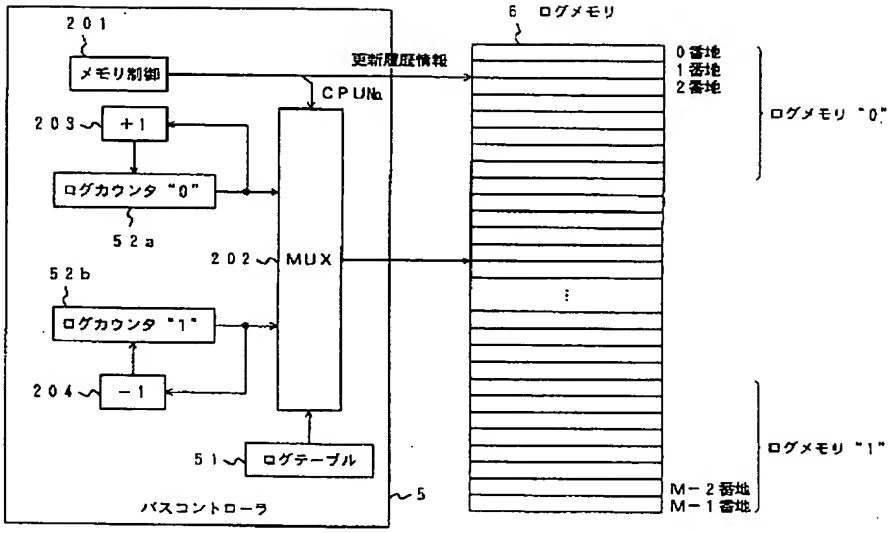
【図１４】


START
更新トランザクションを受取る
S41
メインメモリから更新前のデータを読む
S42
ログテーブルからプロセッサ番号に対応するカレントログメモリ番号を得る
S43
メモリアドレス、更新前データ、カレントログメモリ番号をログカウンタ 303 の指す番地に書き込む
S44
ログカウンタ 303 の値を更新する
S45
END

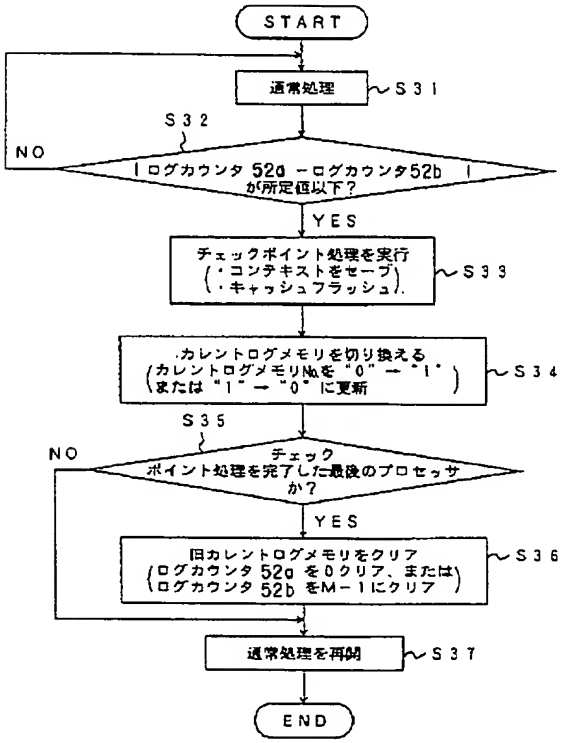
【図７】


(1)
通常処理
(4)
チェックポイント処理
(5)
通常処理
CPU"0"
(ライトバック)
通常処理
通常処理
CPU"1"
(ライトバック)
(4)'
通常処理
CPU"2"
(ライトスルー)
(2)
履歴記録
ログメモリ "0"
520
(6)
履歴記録
ログメモリ "1"
52b
TIME
(3)
ログメモリ "0" の残量が所定容量以下
(チェックポイント処理開始)
(7)
すべてのCPUがチェックポイント処理を終了
(ログメモリ "0" のリセット)

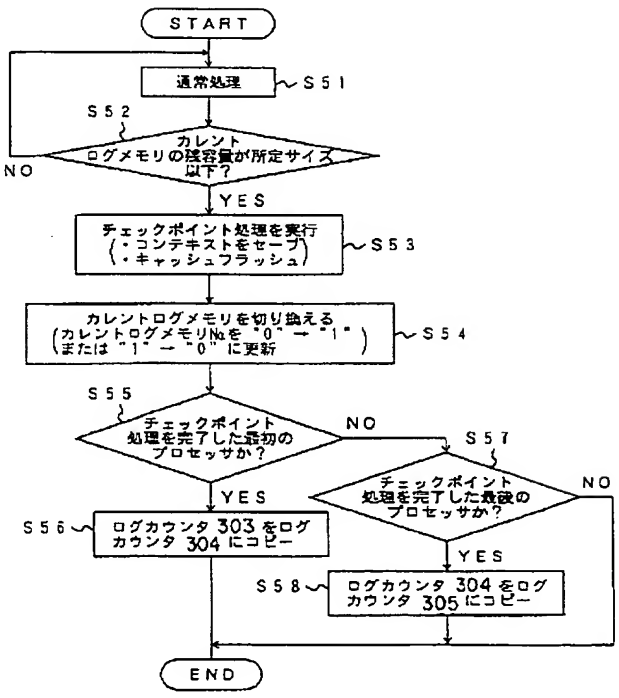
【図９】


201
メモリ制御
更新履歴情報
CPUNo
203
+1
ログカウンタ "0"
52a
202
MUX
52b
ログカウンタ "1"
204
−1
51
ログテーブル
バスコントローラ
5
6 ログメモリ
0番地
1番地
2番地
ログメモリ "0"
ログメモリ "1"
M−2番地
M−1番地

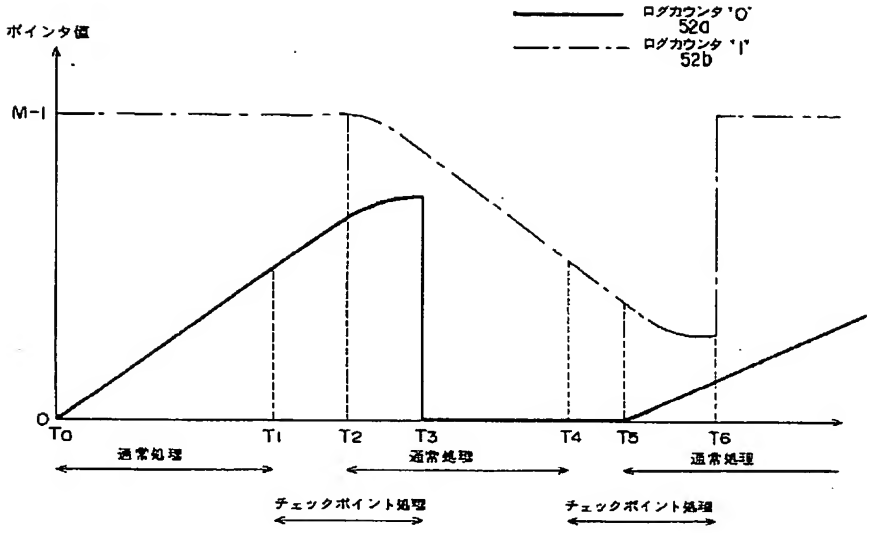
【図１１】


START
通常処理
S31
S32
NO
| ログカウンタ 52a − ログカウンタ52b |
が所定値以下？
YES
チェックポイント処理を実行
（・コンテキストをセーブ）
（・キャッシュフラッシュ）
S33
カレントログメモリを切り換える
（カレントログメモリNoを "0" ⇒ "1"
または "1" ⇒ "0" に更新）
S34
S35
NO
チェック
ポイント処理を完了した最後のプロセッサか？
YES
旧カレントログメモリをクリア
（ログカウンタ 52a を0クリア、または
ログカウンタ 52b をM−1にクリア）
S36
通常処理を再開
S37
END

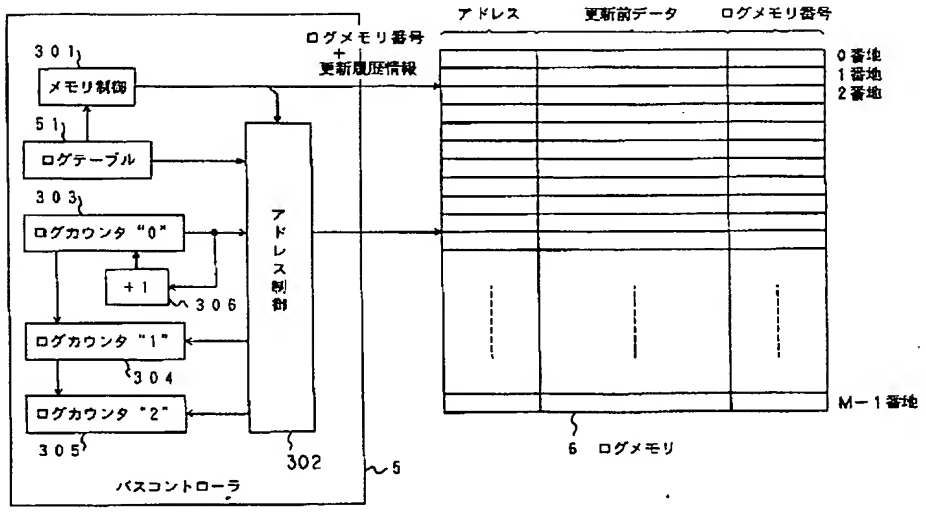
【図１５】


START
通常処理
S51
S52
NO
カレント
ログメモリの残容量が所定サイズ
以下？
YES
チェックポイント処理を実行
（・コンテキストをセーブ）
（・キャッシュフラッシュ）
S53
カレントログメモリを切り換える
（カレントログメモリNoを "0" ⇒ "1"
または "1" ⇒ "0" に更新）
S54
S55
チェックポイント
処理を完了した最初の
プロセッサか？
NO
S57
チェックポイント
処理を完了した最後の
プロセッサか？
NO
YES
S56
ログカウンタ 303 をログ
カウンタ 304 にコピー
YES
S58
ログカウンタ 304 をログ
カウンタ 305 にコピー
END

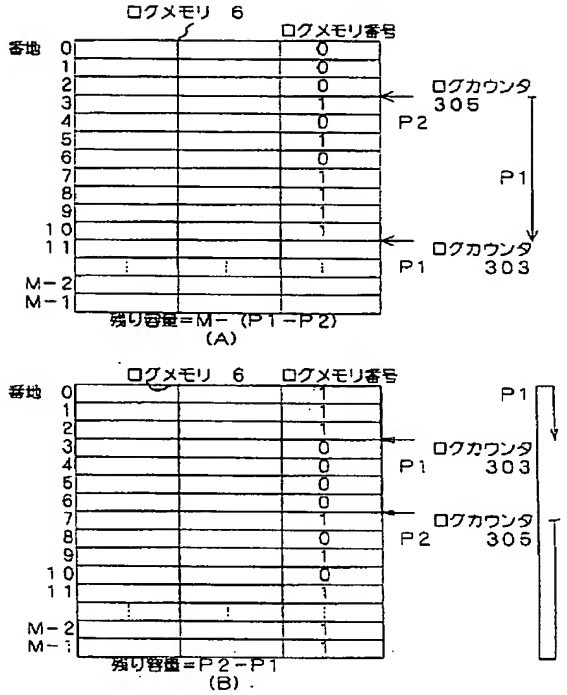
【図１２】


ポインタ値
ログカウンタ "0" 52a
ログカウンタ "1" 52b
M-1
0
T0
T1
T2
T3
T4
T5
T6
通常処理
通常処理
通常処理
チェックポイント処理
チェックポイント処理

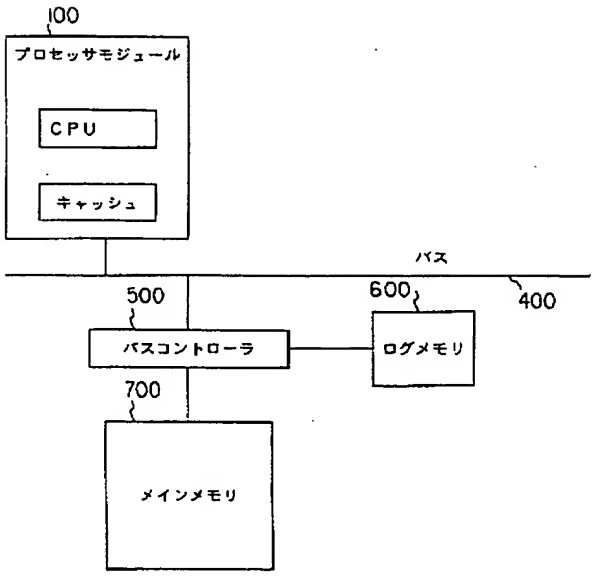
【図１３】


301
メモリ制御
51
ログテーブル
303
ログカウンタ "0"
+1
306
ログカウンタ "1"
304
ログカウンタ "2"
305
アドレス制御
302
5
バスコントローラ
ログメモリ番号
+
更新履歴情報
アドレス
更新前データ
ログメモリ番号
0番地
1番地
2番地
M－1番地
6 ログメモリ

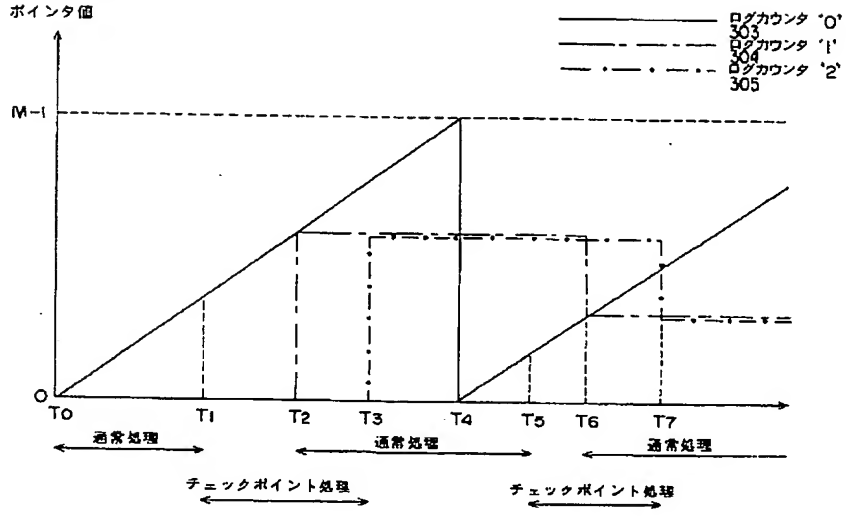
【図１６】


ログメモリ　6
ログメモリ番号
番地
ログカウンタ
305
P2
P1
ログカウンタ
303
残り容量＝M－（P1－P2）
(A)
ログメモリ　6
ログメモリ番号
番地
P1
ログカウンタ
303
ログカウンタ
305
P2
残り容量＝P2－P1
(B)

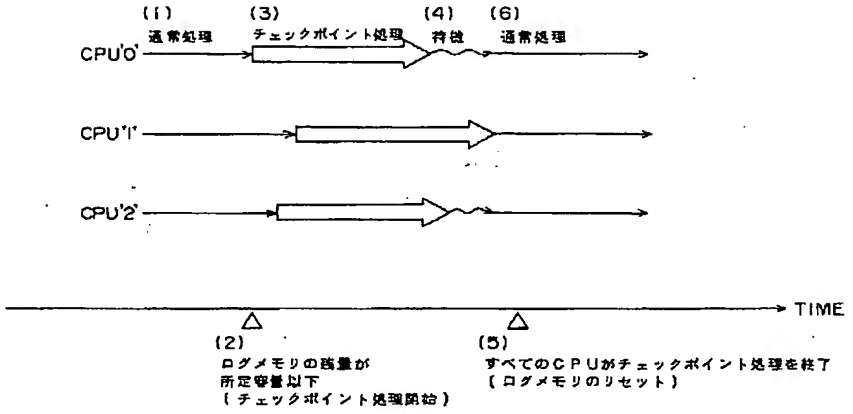
【図１８】


100
プロセッサモジュール
CPU
キャッシュ
バス
500
600
400
バスコントローラ
ログメモリ
700
メインメモリ

【図１７】


ポインタ値
ログカウンタ '0'
303
ログカウンタ '1'
304
ログカウンタ '2'
305
M-1
0
T0
T1
T2
T3
T4
T5
T6
T7
通常処理
通常処理
通常処理
チェックポイント処理
チェックポイント処理

【図１９】



フロントページの続き

(72)発明者　大森　誉史
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

(72)発明者　増渕　美生
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

(72)発明者　藤井　高広
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.